NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A DISTRIBUTED PASSWORD SCHEME FOR NETWORK OPERATING SYSTEMS

by

Christopher Roth

June 2002

Thesis Advisor:	Bret Michael
Co-Advisor:	Craig Rasmussen

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2.REPORT DATE** June 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Title (Mix case letters) A Distributed Password Scheme for Network Operating Systems	**5. FUNDING NUMBERS**
6. AUTHOR(S) **Roth, Christopher**	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT `Approved for public release; distribution is unlimited`	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

Password-based user identification and authentication in a network-based operating system generally relies upon a single file that contains user information and the encoded or hashed representations of each users' password. Operating system designers have resorted to various protection schemes to prevent unauthorized access to this single file. These techniques have proved vulnerable to various attacks, the result being unauthorized access to the targeted computer system. This paper proposes a model for a distributed password system in a network environment that eliminates the single password file as a target without introducing additional computational complexity or incorporating additional cost to the user with such items as tokens or biometrics. This application incorporates proven encryption techniques and a distributed architecture to enhance the reliability of an operating system's identification and authentication procedures. The paper provides an object-oriented model of this approach, along with an analysis of a possible implementation in a current operating system.

14. SUBJECT TERMS Password, Encryption, Attacker, Exploitation, Vulnerabilities, Security	**15. NUMBER OF PAGES** 49
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

DISTRIBUTED PASSWORD SCHEME FOR NETWORK OPERATING SYSTEM

Christopher Roth
Major, United States Army
B.A., La Salle University, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2002

Author: Christopher Roth

Approved by:

James B. Michael
Thesis Advisor

Craig Rasmussen
Co-Advisor

Christopher Eagle
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Password-based user identification and authentication in a network-based operating system generally relies upon a single file that contains user information and the encoded or hashed representations of each users' password. Operating system designers have resorted to various protection schemes to prevent unauthorized access to this single file. These techniques have proved vulnerable to various attacks, the result being unauthorized access to the targeted computer system. This paper proposes a model for a distributed password system in a network environment that eliminates the single password file as a target without introducing additional computational complexity or incorporating additional cost to the user with such items as tokens or biometrics. This application incorporates proven encryption techniques and a distributed architecture to enhance the reliability of an operating system's identification and authentication procedures. The paper provides an object-oriented model of this approach, along with an analysis of a possible implementation in a current operating system.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge Rob "Whitey" Beck, @stake's security engineer. Our conversation on a January afternoon sent me down this path.

The author would also like to acknowledge the support of the C4 Group of the National Security Agency for the direction and resources provided through the Computer Network Research Lab (CNRL).

The author would like to thank Dr. Bret Michael for his vision, guidance, and patience during this research effort. The author would also like to thank Dr. Craig Rasmussen for his critical review of this thesis.

Finally, the author owes a tremendous amount of gratitude to his wife, Noële, his son Patrick, and his daughter Lauren, for their enduring patience, support and love during the completion of this work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

Unauthorized access to an information system is a hacker's dream, a computer user's or owner's headache, and a computer administrator's nightmare. Some of the better-known attacks on computing systems over the past twenty years have relied to some extent on "cracking" password files in order for the attack program to obtain privileged use of the targeted system. If the administrator account, or that of another trusted user, is compromised, then the entire information system may become compromised.

As the sophistication of the techniques for protecting stored passwords has increased, so too have the methods used by adversaries to subvert such protection mechanisms. Encryption has been used to protect stored and transmitted password data. However, examples abound of poorly implemented password encryption schemes, and of the use of password-encryption schemes that are not appropriate (e.g., not strong enough, or too strong) for use with a particular type of information system.

B. DISCUSSION

An operating system bases much of its protection on "knowing" who a user of the system is [Ref. 1]. A valid user needs to be identified. This is usually done with a user identification, or user id. Though there is no standard convention, most systems use a combination of a valid user's names and/or initials. For example, the convention of using the first initial of the first name followed by the full last name would create for this author the user id of "croth." If there are multiple users with similar names, then a middle initial or a series of numbers (e.g. croth, croth2, etc.) might be used to distinguish each of the individuals. Within a given system, a user id must be uniquely associated with only one user.

Once the system is presented with a valid user id, the system must verify that the presenter is truly an authorized user, and not someone masquerading as an authorized user. This is user *authentication.* The authentication process is based on shared knowledge that only the user and the computer would possess. The most common mechanism is a password [Ref. 1].

In response to input of a valid user id, the computer prompts the presenter of the id for the associated password. The computer applies either an encryption algorithm or a hashing algorithm to the password and compares the result to what is stored in the password file. If this value matches the value associated with the user id presented, then the user is granted access to the system. If value does not match, then the user does not gain access to the system.

Note that the passwords themselves are not stored. It is the computed values associated with them that are stored on the system in a password file. For the sake of simplicity, we may view a password file entry as consisting of a user id and the hashed or encrypted image of the user's password. The schemes by which these password files are stored mighty vary by system, but they all contain the same data: a list of user ids together with their associated password values.

C. HYBRID ENCRYPTION SCHEME: A WAY FORWARD

In this thesis we introduce a hybrid encryption scheme that involves distributing the protection throughout the physical components of the information system, obviating the need for centralized storage of password data. The scheme will use various encryption techniques in conjunction with the distributed protection to eliminate a single point of failure, or (in an adversary's case) a single "golden target": the password file. The scheme attempts to mitigate the risks to a systems password file from documented failures, as well as weaknesses that have been exploited, by building a more secure identification and authentication (I&A) process without incurring the additional costs of smart cards, tokens, or the incorporation of biometrics.

II. EXPLOITATION OF PASSWORDS AND PASSWORD FILES

After 11 September 2002, the United States has become more aware of its vulnerabilities both in the physical world and in the cyber world. Recently a *USA Today* newspaper article reported:

> The vast array of potential targets and the lack of adequate safeguards have made addressing the threat daunting. Among the recent targets that terrorists have discussed, according to people with knowledge of intelligence briefings:
>
> - The Centers for Disease Control and Prevention, based in Atlanta. It is charged with developing the nation's response to potential attacks involving biological warfare.
> - The nation's financial network, which could shut down the flow of banking data. The attack would focus on the FedWire, the money-movement clearing system maintained by the Federal Reserve Board.
> - Computer systems that operate water-treatment plants, which could contaminate water supplies.
> - Computer networks that run electrical grids and dams.
> - As many targets as possible in a major city. Los Angeles and San Francisco have been mentioned by terrorists, intelligence officials say.
> - Facilities that control the flow of information over the Internet. Richard Clarke, the White House special adviser on cybersecurity, says such sites, of which there are 20 to 25, are "only secure in their obscurity."
> - The nation's communications network, including telephone and 911 call centers.
> - Air traffic control, rail and public transportation systems.
>
> Officials are most concerned that a cyberattack could be coupled with a conventional terrorist attack, such as those on Sept. 11, and hinder rescue efforts [Ref. 2].

A. THE BASIS OF THE CONCERNS

The terrorist attacks on the World Trade Center and the Pentagon, as well as the Oklahoma City bombing, showed that high-profile targets are vulnerable. They also demonstrated that it is not hard to obtain the necessary weapons and training to carry out such attacks on United States (U.S.) soil. Although physical terrorist attacks on U.S. soil are relatively new, attacks on computer systems throughout the U.S. have occurred since computers became capable of communicating with one another.

1. Infomaster and the Penetration of Bureau of Land Management

In the spring of 1992, "Infomaster", a 'hacker' of limited skills but enormous persistence, had remotely penetrated the computer systems of the Bureau of Land Management (BLM) in Portland, Oregon [Ref. 3]. From there, he had access to the agency's national network, which included the BLM office in Sacramento. The computers in the Sacramento office controlled every dam in the northern part of the state [Ref. 3]. With a few simple commands, the attacker "could bury some of the world's richest agriculture land beneath a tidal wave, killing hundreds of people, destroying thousands of homes, and throwing the futures (commodities) market into chaos." [Ref. 3]. Infomaster used basic techniques to gain access to the BLM and other networks. He guessed passwords. He penetrated and altered password files. He used tools available to "crack" passwords [Ref. 3].

2. The INTERNET Worm of 1987

Even Robert T. Morris, the author of the INTERNET Worm that brought the infant INTERNET to a stop in 1987, used the weaknesses of passwords and password files in conjunction with the vulnerabilities in the *sendmail* program and the *finger* program [Ref. 1, 4]. The worm tried guessing passwords. When it succeeded, it penetrated the system and captured the password file. The password file contained the passwords in encrypted form, but the ciphertext of each password was visible. Morris' worm encrypted various popular passwords and compared the resulting ciphertext to the entries in the password file [Ref. 1, 4]. If unsuccessful, the worm tried the dictionary file stored on the system for use by spelling checkers [Ref. 1, 4]. Whenever it got a match, it would log into the account and then start looking for other machines to attack [Ref. 1, 4]. Morris' worm was classified as benign, in that it collected the account passwords but did not save them [Ref. 1].

3. The First Documented Case of Cyber Espionage

As early as 1988, Clifford Stoll became the author of the first book documenting a case of cyber espionage. As Stoll documented his case, he discovered that passwords and password files became two of the main targets [Ref. 5]. Stoll “witnessed” the attacker editing password files, deleting the passwords from old users, and basically bringing old, inactive accounts to life [Ref. 5]. The attacker did not break the encryption, which at the time was done using the Data Encryption Standard (DES), but actually stole accounts by deleting or changing the passwords that were in the password file [Ref. 5].

4. Password Insecurities of Tomorrow

Though all these cases might seem dated to today’s more sophisticated attacks, passwords and password files are still prime targets of attackers. It was recently discovered that, as Microsoft shifts its focus and strategy to its new .Net framework of integrated web-based software delivery, the Microsoft Developer Network documentation instructs developers to create a file containing users’ passwords and place it in a directory accessible from the Web, providing a potentially lucrative target for attackers [Ref. 6].

B. PRIMARY TARGET

All of the actual cyber attacks illustrated above occurred when passwords and password files were compromised to give an attacker access to a system. Once a system is penetrated, the exploitation of the password file allows the attacker further penetration, and escalation of rights and privileges. The password file is a single target, which can become a single point of failure.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PASSWORD SCHEMES

A. MICROSOFT NETWORKING OPERATING SYSTEMS

In Microsoft's network operating systems, Windows NT and Windows 2000, the passwords are not stored in clear text. Windows NT stores user-related information in the System's Account Manager (SAM) portion of the domain controllers' registries. The SAM does not store the actual passwords, but stores two 16-byte hash values of the password [Ref. 7].

1. Login and Authentication

The passwords are never exchanged between the client and the server, either. The NT network uses a challenge-and-response system, called Challenge Handshake Authentication Protocol (CHAP) [Ref. 7]. As a user logs in as a client, the client side calculates a 16-byte hash value of the user's password [Ref. 7]. The client then connects to the server and sends the user id across the network [Ref. 7]. The server generates a random eight-byte nonce, known as the challenge, and sends it to the client [Ref. 7]. The client uses three distinct DES keys to encrypt the challenge. Key one contains the first seven bytes of the password's hash value [Ref. 7]. Key two contains the next seven bytes in the password's hash. Key three contains the remaining two bytes of the password's hash, to which are appended five zero filled bytes. The client system applies each key to the nonce so that the eight-byte challenge results in three 64-bit outputs, which is the 24-byte response [Ref. 7]. After the server generates a nonce and sends the challenge, it looks up the user's password hash value stored in the SAM database [Ref. 7]. The server then creates a response by performing the same calculation that the client performed using the nonce and hash value. If the responses match, then the server authenticates the user (figure 3.1) [Ref. 7].

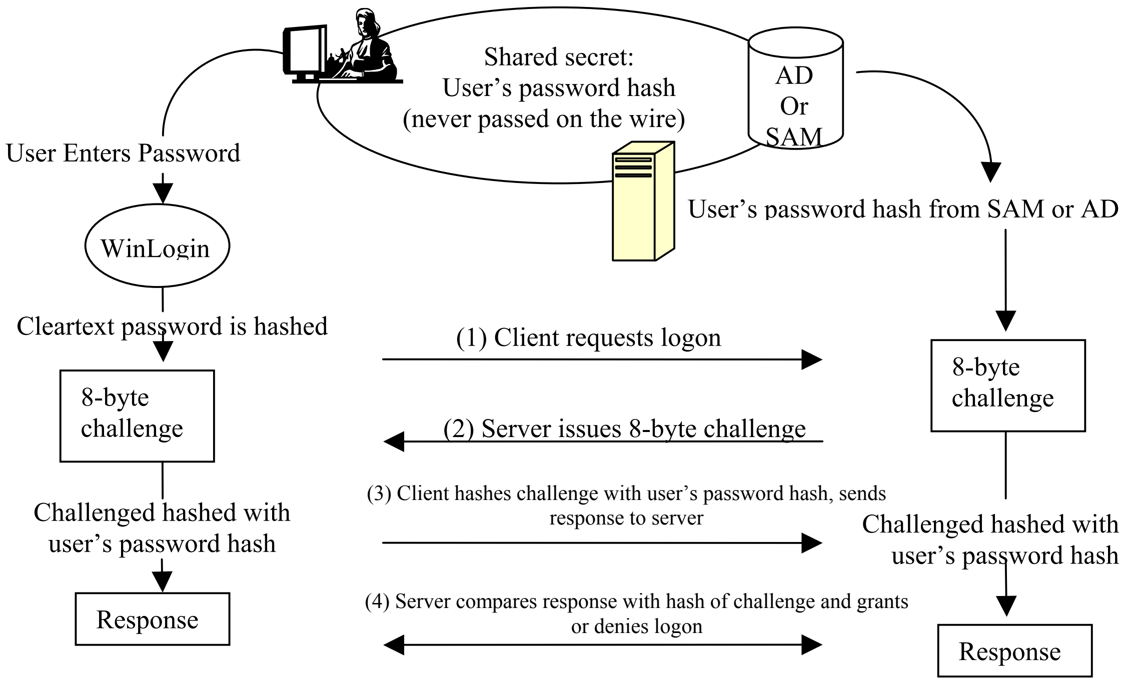

Shared secret:
User's password hash
(never passed on the wire)
AD
Or
SAM
User Enters Password
WinLogin
User's password hash from SAM or AD
Cleartext password is hashed
(1) Client requests logon
8-byte challenge
8-byte challenge
(2) Server issues 8-byte challenge
(3) Client hashes challenge with user's password hash, sends response to server
Challenged hashed with user's password hash
Challenged hashed with user's password hash
(4) Server compares response with hash of challenge and grants or denies logon
Response
Response

Figure 3.1 Challenge/Response Authentication [Ref. 7]

2. Password Storage

The SAM file stores the password information using a one-way hashing algorithm. Theoretically speaking, this is a function *h* which is easy to compute, but for which it is computationally infeasible to find two messages *M* and *M'* such that *h(M) = h(M')*. [Ref. 8]. In other words, once the password is hashed using this function, the value cannot be decrypted by any practical method. For all NT systems and Windows 2000 stand-alone systems, the SAM file is kept in the file %systemroot%\system32\config\sam [Ref. 7]. On Windows 2000 domain controllers, this information is kept in the Active Directory (%systemroot%\ntds\ntds.dit) [Ref. 7]. The format for the SAM files is the same in either case, but they are accessed differently [Ref. 7].

3. System Key (SYSKEY)

Later versions of NT (NT4 Service Pack 3) and Windows 2000 provided more security with an additional layer of encryption. This additional layer is the System Key, or SYSKEY. Once the hashes are computed, the SYSKEY then encrypts the hashes, using a random 128-bit key. The SYSKEY can be stored in three ways: in the registry and available automatically upon boot-up (the default), in the registry but locked with a password that must be provided at boot time, or stored on a floppy disk that must be supplied at boot time [Ref. 7].

4. Single Password File

The final layer of protection is the SAM file and its content can only be accessed with Administrator privileges. The bottom line is that, although it is protected, there still remains a single password file.

B. UNIX

Since there are multiple versions of the Unix operating system, we will discuss the generic password scheme.

1. Password File Storage

Like Windows, Unix does not store passwords themselves in a password file, but stores the encrypted password along with some additional information. A user types in a password of up to eight characters. This is converted into a 56-bit value (using seven-bit ASCII) that serves as the key input into the encryption routine. The encryption routine, known as crypt (3) is based on DES. The DES algorithm is modified using a 12 bit "salt" value, which is usually tied to the value of the computer's system clock at the time when the password was assigned to the user. This modified algorithm is exercised with a data input consisting of 64-bit block of zeros. The output is then used as the input for a second encryption. The process is repeated for a total of 25 encryptions. The resulting 64-bit encryption is then translated into an 11-character sequence [Ref. 9, 10].

This result, along with a plaintext copy of the salt, is stored in the password file (figure 3.2). The salt assists in the prevention of duplicate copies of the encrypted password. The salt is tied to the time of password creation, then attached to the password before it goes through the encryption routine. The chance that two users, with the same passwords, have the same encrypted value is one in 4096. [Ref .10]. The salt also increases the size of the password without any additional burden on the user, and it prevents the use of a hardware implementation of DES to assist in a brute force guessing attack [Ref. 9].

2. Crypt (3) Function Key move to next page

The key used for the crypt (3) function is the user's password; the actual encoded string is all nulls. A point of clarification is needed. Though most literature calls this encryption, cryptographers call this encoding, since the encoded string is all null. The salt is a two-character string chosen from the set [a-zA-Z0-9./], linked to the time at which the password was first created. This allows the algorithm to be perturbed in one of 4096 ways [Ref. 11].

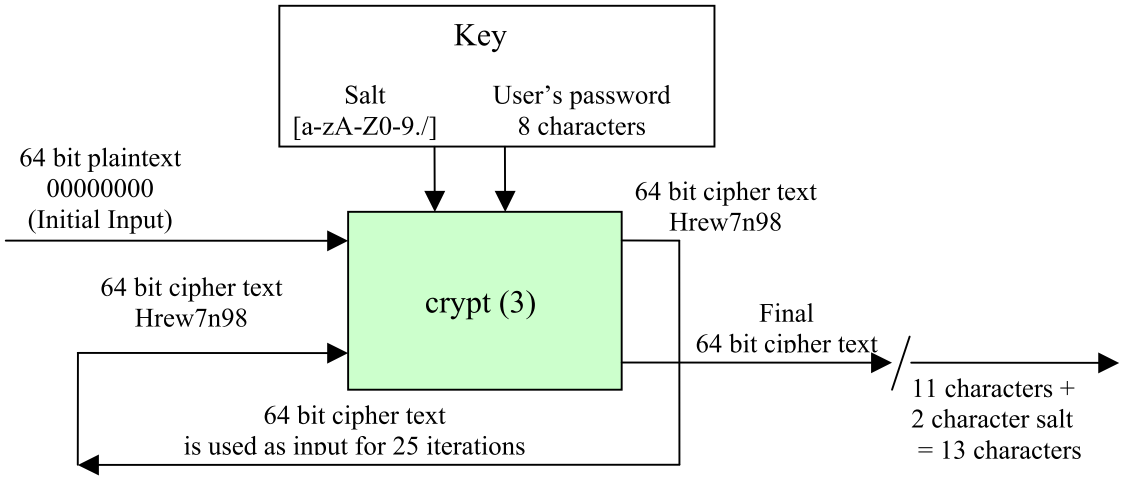


Figure 3.2 UNIX Password Implementation.

3. User login

When a user logs on to a Unix system, the user provides a unique user ID and a password. The operating system then uses the user ID to index into the password file and retrieve the plaintext salt value and the encrypted password; these are used as input to the encryption routine. If the result matches the stored value, the password is accepted [Ref. 9].

The eight-character password is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to the encryption routing. This 56-bit value allows the key space to consist of 2^{56} possible values.

4. System Storage of the Password File

Unix stores the password file in the */etc/passwd* file. This file is world-readable, meaning anyone can have access to the file. Some versions, like LINUX, provide added protection by shadowing the password file. This relocates the password file's values to another file (*/etc/shadow*) so that it can be read only by a user with root privileges. The original password file */etc/passwd* no longer contains the encrypted values; it just contains an x value that indicates that the password files have been shadowed. The encrypted passwords have not been changed or modified; they have just been moved to a more protective file.

The shadow suite also adds additional security features, such as tracking password changes, age of password. It also allows for the use of longer than eight character passwords. Even with all the additional security features added, the system is still left with a single file that holds user id and encryption. [Ref. 10].

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PASSWORD IMPLEMENTATION

A. WEAKNESSES OF PASSWORDS

A user is identified by a unique user id and authenticated with a password. The password should, in theory, be easy to remember, hard to guess, and only known to that user. In actuality, passwords are one of the weaknesses of computer security.

1. Password Attacks

Passwords can be attacked in several different manners. An attacker could try all possible password combinations. He could try many probable passwords. Knowing the user might provide clues to the user's password. The system, or even the work area might contain the password, in a written or electronic form. If all else fails, an attacker might just ask the user for the password. Foundstone, Inc., a computer security consulting and training organization, states, "Weak passwords are the primary way in which we defeat Windows 2000 networks in professional penetration testing engagements." [Ref. 7].

2. Poor Password Choices

Various studies from the late 1970s to today have shown that users tend to choose poor, easily guessable, or swiftly cracked passwords. What's worse, the studies show that users do not learn from previous mistakes or examples. In 1979, a sample consisting of 3,300 passwords indicated that, given a reasonable amount of time using the tools available, eighty-six percent could be uncovered in one week. [Ref. 1, 12] (Table 4.1).

Actual Number	Percent	Description
15	.5 %	single ASCII character
72	2%	two ASCII character
464	14%	three ASCII character
477	14%	four alphabetic character
706	21%	five alphabetic characters, all of the same case
605	18%	six lowercase alphabetic characters
492	15%	words in dictionaries or lists of names
2831	86%	Total of all above categories

Table 4.1 Distribution of Actual Passwords. From [Ref. 1].

The results of similar studies conducted in 1990 and 1992, in which five times as many passwords were collected, showed that the same problems were still occurring, even after Morris' Internet Worm of 1988 used these weaknesses to spread throughout the Internet, and eventually bring it to its knees [Ref. 1].

3. Policies to Protect Passwords

As system administrators and designers became more educated in how an attacker can gain access to a system through weak passwords, they established criteria for "good" passwords. These were both written and computer enforced policies. They included increasing the character space to include, not just letters and numbers but also special characters, changing the case at least once, making the password longer, avoidance of actual names or words, use of an unlikely password, regular replacement of the password and the strong recommendation that passwords should neither be written down nor shared [Ref. 1].

A password of length of three characters or less of a single case can be any one of $26 + 26^2 + 26^3 = 18{,}278$ possible combinations. Using an assumed rate of one password per millisecond, every combination could be tried in 18.278 seconds. Even increasing the character length to four or five increases the time to approximately eight minutes or three and one-half hours, respectively. Pfleeger states that it would take one hundred hours to test all six-letter words from letters of only one case, but it would take approximately two years to test all six-symbol passwords from the set of all upper- and lower-case letters and all decimal digits. Using single-case letters, there are 26^6 possible six-character passwords. Searching for all possible combinations in a standard English collegiate dictionary, 99.95% of these would not be found [Ref. 1].

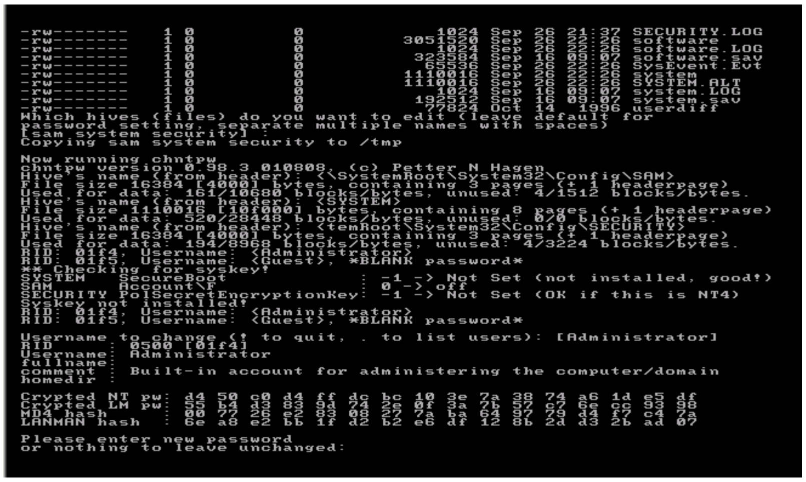
Even with the advent of these policies, the implementation of passwords yielded additional vulnerabilities that could be exploited. Some of these vulnerabilities were caused by users who managed to adhere to the letter of the policies as opposed to the spirit of the policies. Others were based on flawed implementations produced by the manufacturers of the systems.

B. WINDOWS FLAWS

The SYSKEY was added to later versions of NT and implemented in WINDOWS 2000. The SYSKEY provides another layer of protection of the password file. It actually takes the hashes in the password file and further encrypts them. This was to prevent a brute force attack or dictionary attack. As stated previously, the SYSKEY could be stored several different ways, and added another layer of protection. SYSKEY applies a second, 128-bit strong round of encryption to the password hashes using a unique key that is either stored in the registry, optionally protected by a password, or on a floppy disk [Ref. 7, 13]. However, these measures provide a false sense of security due to backward compatibility issues.

1. Old Hash Conversion

A user can actually inject fraudulent hashes and bypass WINDOWS' security features. Petter Nordhal-Hagen actually developed a tool that allows an attacker who has physical control of the box to boot a WINDOWS box into a LINUX operating system. The tool pulls the SAM file into a temporary directory, and then allows an attacker to change the password of any user. It does this by using Microsoft's hashing algorithm. It then places the new hashes into the SAM file, and writes the new SAM file back into the system. Nordhal-Hagen also discovered that if the SYSKEY is enabled, it will automatically convert the old-style hashes (without the SYSKEY's encryption) to the new SYSKEY'ed hashes once the system is rebooted [Ref. 7] (Figure 4.1). The WINDOWS security features do not log the changes because they are done in a different operating system. This attack will not work with the WIN2K domain controllers, because they store the password hashes in the active directory and not in the SAM file. But a more refined technique, to which the WIN2K domain controllers are susceptible, might not be far off [Ref. 7, 13].



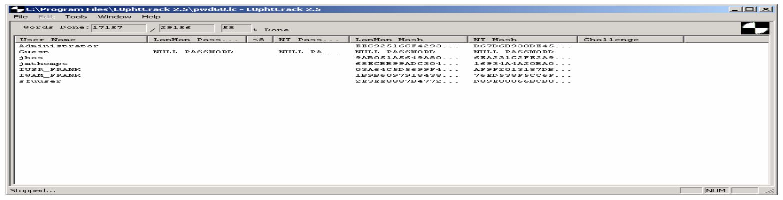
```
-rw-------   1 0        0          1024 Sep 26 21:37 SECURITY.LOG
-rw-------   1 0        0       3051520 Sep 26 22:26 software
-rw-------   1 0        0          1024 Sep 26 22:26 software.LOG
-rw-------   1 0        0        323584 Sep 16 09:07 software.sav
-rw-------   1 0        0         65536 Sep 26 22:26 SysEvent.Evt
-rw-------   1 0        0       1110016 Sep 26 22:26 system
-rw-------   1 0        0       1110016 Sep 26 22:26 SYSTEM.ALT
-rw-------   1 0        0          1024 Sep 16 09:07 system.LOG
-rw-------   1 0        0        192512 Sep 16 09:07 system.sav
-rw-------   1 0        0         77824 Oct 14  1996 userdiff
Which hives (files) do you want to edit (leave default for
password setting, separate multiple names with spaces)
[sam system security] :
Copying sam system security to /tmp

Now running chntpw
chntpw version 0.98.3 010808, (c) Petter N Hagen
Hive's name (from header): <\SystemRoot\System32\Config\SAM>
File size 16384 [4000] bytes, containing 3 pages (+ 1 headerpage)
Used for data: 161/10680 blocks/bytes, unused: 4/1512 blocks/bytes.
Hive's name (from header): <SYSTEM>
File size 1110016 [10f000] bytes, containing 8 pages (+ 1 headerpage)
Used for data: 520/28448 blocks/bytes, unused: 0/0 blocks/bytes.
Hive's name (from header): <temRoot\System32\Config\SECURITY>
File size 16384 [4000] bytes, containing 3 pages (+ 1 headerpage)
Used for data: 194/8968 blocks/bytes, unused: 4/3224 blocks/bytes.
RID: 01f4, Username: <Administrator>
RID: 01f5, Username: <Guest>, *BLANK password*
** Checking for syskey!
SYSTEM   SecureBoot                : -1 -> Not Set (not installed, good!)
SAM      Account\F                 : 0 -> off
SECURITY PolSecretEncryptionKey: -1 -> Not Set (OK if this is NT4)
Syskey not installed!
RID: 01f4, Username: <Administrator>
RID: 01f5, Username: <Guest>, *BLANK password*

Username to change (! to quit, . to list users): [Administrator]
RID     : 0500 [01f4]
Username: Administrator
fullname:
comment : Built-in account for administering the computer/domain
homedir :

Crypted NT pw: d4 50 c0 d4 ff dc bc 10 3e 7a 38 74 a6 1d e5 df
Crypted LM pw: 55 b4 d3 83 9d 74 2e 0f 3a 7b 57 c7 6e cc 93 98
MD4 hash     : 00 77 26 e2 83 08 27 7a ba 64 97 79 d4 f7 c4 7a
LANMAN hash  : 6e a8 e2 bb 1f d2 b2 e6 df 12 8b 2d d3 2b ad 07

Please enter new password
or nothing to leave unchanged:
```

Figure 4.1 Nordhal-Hagen WIN2K/NT Password Recovery Tool

There are versions of the utility that boot the system into MS-DOS, and then change the passwords by changing the hashes. Again, the hashes are never "cracked;" they are just rewritten.

2. Password "Cracking"

Even though there are no known mechanism for decrypting the passwords hashed using the NT/2000 algorithms, password are recovered from the hashes using various tools [Ref. 7]. L0phtcrack and John the Ripper are just a few of the tools that duplicate the hashing techniques that WINDOWS uses to match hashes stored in the SAM file [Ref. 7]. What makes it even easier to match the hashes is another flaw in the Microsoft networking operating system in order to make it backward compatible with its predecessors (figure 4.2).


C:\Program Files\L0phtCrack 2.5\pwd68.lc - L0phtCrack 2.5
File Edit Tools Window Help

Words Done: 17157 / 29156 58 % Done

User Name	LanMan Pass...	<8	NT Pass...	LanMan Hash	NT Hash	Challenge
Administrator				EEC92516CF4293...	D67D6B930DE45...	
Guest	NULL PASSWORD		NULL PA...	NULL PASSWORD	NULL PASSWORD	
jbos				9AB051A5649A80...	6EA231C2FE2A9...	
jmthomps				68ECBB99ADC304...	16934A4A20BA0...	
IUSR_FRANK				03A64C5D5699F4...	AF9F2013187DB...	
IWAM_FRANK				1B9B6097918438...	76ED538F5CC6F...	
sfuuser				2E3EE8887B4772...	D89E00066BCB0...	

Stopped...

Figure 4.2 L0phtCrack 2.5

a. Backward Compatibility

(1.) Local Area Network Manager (LANMan). This backward compatibility issue has come back to haunt Microsoft's network operating system. It is the implementation of the Local Area Network (LAN) Manager hash. This is a key design failing of Windows NT/2000 [Ref. 7, 13]. Both NT and WIN2000 store two versions of hashes for a user: the LANManager (LM) hash and the NT LANManager (NTLM) hash.

The first eight bytes of the LM hash are derived from the first seven characters of the user's password, and the second 8 bytes are derived through the eighth through the fourteenth character [Ref. 7, 13]. An eight-character password actually reduces to a seven-character password together with a one-character password (figure 4.3). Searching the space of seven-character strings is not difficult with modern computers. Both tools mentioned automate this process, making it very easy to match the hashes [Ref. 7, 13, 14].

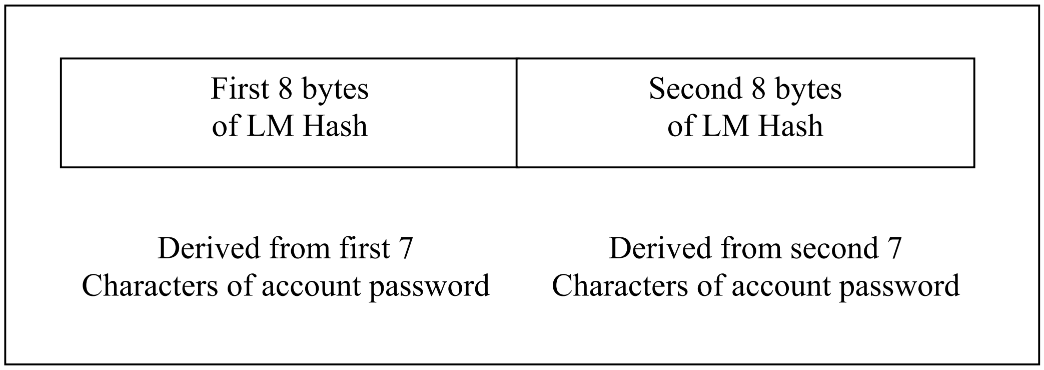

First 8 bytes of LM Hash
Second 8 bytes of LM Hash
Derived from first 7 Characters of account password
Derived from second 7 Characters of account password

Figure 4.3 LanMAN Hash [Ref. 7].

The system administrator does have the ability to turn off the storage of the LM hash, but there are consequences. This can break certain applications, and is only recommended on test systems, not on actual production systems [Ref. 7]. Even by disabling the LM storage, currently-stored LM hashes are not erased [Ref. 7]. The only way to prevent the need for an LM hash for authentication is to have a strictly homogenous Windows 2000 environment using the built-in Kerberos v5 protocol that is new in Windows 2000 [Ref. 7]. This is not a default setting, while the LM hash is, and there is currently no mechanism to force the use of Kerberos [Ref. 7].

(2..) Local Security Policy Setting Store. Another compatibility issue is in the Local Security Policy Setting Store Passwords with Reversible Encryption. Though this is only applicable on the Active Directory Domain Controllers, it does lead to the ability of passwords being stored in a reversible encryption, instead of a one-way hash. By default it is turned off, but, if the Domain Controller is compromised by an attacker, this setting can be enabled. This forces all newly created passwords to be stored in the SAM/AD form as normal, and also in a separate reversible encrypted format [Ref. 7]. Microsoft uses this with remote protocols and services like MSChap v1, Digest Authentication, Apple Talk Remote Access, and Internet Authentication Services, all of which require this setting. Although there is no tool that could dump the plaintext passwords while Reversible Encryption is enabled, the ability to create one does exist [Ref. 7].

C. UNIX FLAWS

Older versions of UNIX, and the modern UNIX-flavor operating systems, use DES for their encryption. DES is limited to eight characters as the key, so in actuality a user is limited to only an eight-character password. Newer versions of the software have replaced DES with the MD5 hashing algorithm. This has improved on DES in several ways. The key is no longer limited to eight characters. In actuality the passwords could have infinite length. The MD5 keyspace is larger than DES.

1. World Readable Password File

Even with the switch to from DES to MD5, the password file is still world-readable. Having a password file world-readable allows any user to grab it, take the hashes, and run them through a password cracker. Current versions of Linux add the additional protection of "shadowing" the password file. The shadow password file contains the encrypted or hashed versions of the passwords on the system and makes them readable by root. Shadowing is considered essential for password security [Ref. 11]. There are packages that allow for modification of older Unix operating systems, to create a shadow file. At the same time, there are current versions of UNIX that are still not using the shadow capability. The most surprising is that MAC OS X, built on Open BSD, has a utility called NetInfo that "has a 'feature' that, strangely, gives out the hashed passwords to ANY user that is logged on (not sure if this is because it isn't using a shadow password file, or if NetInfo just plain compromises the shadow). There is a utility called 'Malevolence' that will allow a user to view the password file." [Ref. 15].

2. Shadow Password File Vulnerability

Even shadow passwords and shadow password files are not totally secure. Carolyn Mienel has documented several attacks that work on the shadow file on many, but not all UNIX systems. One attack involves creating a program that makes successive calls to the *getpwent( )* to obtain the password file. There are sometimes backup shadow password files that are readable. Even a core dump or segmentation fault that occurs while running a program that must access a password will contain the password. Searching through the core dump will provide both user name and password [Ref. 16].

Even with all the additional protection of a shadow password, if an attacker manages to gain root access, he can access the shadow password file. The shadow password file could then be run through a password cracker for Unix-flavor systems, such as Crack or John the Ripper. Crack was written with the versatility to use either DES or MD5 for the crypt function. There is also a password cracker called Slurpie that runs in a distributed environment, using multiple machines. This decreases the time needed to crack the password file [Ref. 11].

Unix use of passwords leaves a password exposed in various states and utilities throughout the system. If not properly configured, the password can be relatively unprotected and be retrieved by an attacker. For example, if the utility for downloading email from a remote server is run in a daemon mode, instead of user running it individually, the utility will look for the user's password in its control file. The passwords are stored in this file in clear text. The challenge and response authentication for Post Office Protocol version 3 (POP3), known as Popauth, requires that the server have access to the user's clear text-password. Popauth must store the user's POP3 password in a separate database. This causes two problems. The systems and the POP3 password databases must be synchronized with one another. The second, who is more serious, is Popauth stores the password in its database using a reversible encryption. This file is relatively easy to compromise. What is worse is that the compromise of this file means the system's password file is compromised [Ref. 11].

D. GENERAL WEAKNESSES OF PASSWORDS

Passwords are "cracked" using three methods. These are attacks on password files, dictionary attacks, and the brute force method.

Both the length of the password (the number of alphanumeric characters) and the set of permissible characters (letters, upper and lower case, numbers, additional symbols (i.e.,!, @, #, $, %, ^, &, *, etc.) determine how fast a password can be "cracked." For example, using a password that is four characters in length and consists of upper and lower case letters (alphabet size 52), there are 52^4 possible passwords. Given a password-cracking program that can generate, encrypt and compare 10^6 strings per minute, it will take approximately seven minutes to try all four-character strings. As we increase the alphabet size as well as the length of the password, the time to crack a password by brute force would increase. Given a password of exactly eight characters in length, using the same alphabet size, the number of possible passwords has increased to 52^8. Using the original password-cracking program, given the same hypothetical 10^6 strings per minute, it would take approximately fifty years, on average, to determine the password. The password cracking programs that are currently available, both commercial and freeware, can run faster than this, due to faster processors, updated software, and more efficient coding of attack tools [Ref. 1]. Though no formal statistical analysis has been done, in 1999, Jim Williams, running L0phtcrack 2.5 on a Pentium 166, cracked three-letter passwords in seconds and six-letter passwords in seven minutes, using a dictionary attack. By allowing the attack to run overnight, he was able to crack seven and eight character alphanumeric passwords in less than eight hours. More recently, @Stake has released L0phtcrack version 3.0, and today's processors are faster by an order of magnitude than those discussed here [Ref. 17].

Even L0pht Heavy Industries' engineers have posted some startling statistics. During an audit that they performed of a large high-technology company, using their older version of the software, 90 percent of the passwords were cracked in under 48 hours on a Pentium II/300. Eighteen percent of these passwords were cracked in less than 10 minutes. More importantly, the Administrator and most of the Domain Admin passwords were cracked even though the company had a policy requiring passwords longer than eight characters with at least one upper case character plus a numeric or symbol character [Ref. 17].

These statistics are all based on the brute-force method of trying every combination possible. Using a dictionary attack, or what L0phtcrack calls a "Hybrid" attack, in which dictionary words are mixed with other characters (e.g., 1banana2), the time required would not be as long. This attack is based on the assumption that a user would pick a word that is contained in a dictionary file used in the attack.

As previous stated, the password file is one of the primary targets of an attacker. Knowing this, we want to eliminate the target. Passwords have been a part of the I&A process from the beginning. They have also been one of the Achilles' heels of computer systems. The problems associated with passwords generally result from their being too simple, or too common, to too personal (e.g., containing birth dates, or names of children). Attackers understand this and have created a methodology for testing passwords based on user information, as well as tools for breaking passwords.

As security professionals have come to understand this, more stringent password selection guidelines have been implemented with some success. However, as recently as April 2002, studies have shown that approximately fifty percent of computer users base passwords on the name of a family member, and about thirty percent use a public figure such as a sporting hero or a media idol. Based on the study conducted by Pentasafe Security Technologies Ltd., psychologists at City University in London stated that it is possible to predict passwords based on the personality of the user, or even on the items on a users desk [Ref. 18].

E. IMPLEMENTATION OF PASSPHRASES

Knowing the weaknesses of passwords, computer security experts often advocate the expansion of the password to a passphrase. Passphrases meet three security goals of the security-oriented network administrator:

1. They are easy to remember.
2. They are difficult to guess or crack (or at least harder than passwords), by virtue of their greater length.
3. They are inexpensive to implement.

Given a passphrase, one can modify it in various ways. This modification should be chosen to remain easy to remember but more difficult to crack [Ref. 17, 19]. Security experts recommend setting the default length of passwords to the longest available to the system [Ref. 17, 18, 19].

Various programs have implemented passphrases instead of passwords. Pretty Good Privacy (PGP) uses passphrases to generate private keys for public-key cryptography. The forty-character passphrase is converted into a random key using a one-way hashing function. This technique is called key crunching. It creates a pseudo-random bit string, which is used as a key for encryption.

If increasing the length of passwords makes them harder to "crack", why not just replace passwords with passphrases throughout all computer systems for Identification and Authentication?

Passphrases that are English words are relatively weak out to a length of twenty characters. The reason is that, for long streams of text, the redundancy in English is such that each of the 26 letters is comparable to 1.5 to 2.3 bits (as opposed to 4.7 bits if all letters were equally likely) [Ref. 14]. This means that breaking a 15-character passphrase is effectively impossible if the passphrase is randomly chosen from all character combinations, while it is relatively simple if the password is known to be an English phrase [Ref. 14].

The modified passphrases, either encoded or hashed, would be stored in a single file. Security experts still expect password cracking programs to uncover passphrases after sustained effort [Ref. 17]. The weaknesses of the passphrases with the known encryption schemes contribute to this. If additional factors that contribute to cracking passphrases are factored in, a forty-character passphrase may not be as formidable as originally thought. Cryptographers have asserted that searching through forty-character phrases is actually easier than searching through 64-bit random keys [Ref. 20]. A variety of techniques are employed, including Markov chains, phonetic generation algorithms, and concatenation of small words, in the cracking of passphrases [Ref. 14].

Hackers have even attacked PGP's forty-character passphrases. PGPCrack is a widely distributed brute-force utility, designed for cracking conventionally PGP-encrypted files and attacking the secret key's passphrase. PGPCrack relies on a dictionary file, trying each word as a potential passphrase. On a conventionally encrypted PGP file, the utility cycled through over 15,000 words per second on a 100 MHz Pentium. As a point of reference, compare this to the 5,000 to 7,000 words per second tested by typical UNIX password cracking utilities on the same machine [Ref. 21].

Passphrases add to the protection against a straight dictionary attack on a password file. In this context, the advantage of a passphrase is that it is a concatenation of multiple words, and will therefore not be found in any conventional dictionary. An additional benefit, due to a passphrases' length, is that an attacker's attempt to guess an authorized user's password is likely to fail.

THIS PAGE INTENTIONALLY LEFT BLANK

V. A DISTRIBUTED PASSWORD SCHEME FOR NETWORK OPERATING SYSTEMS

The Distributed Password Scheme (DPS) consists of three parts. These are the login use of a passphrase, the segmentation of the passphrase, and the encryption and storage of these segments.

A. MIGRATION FROM PASSWORDS TO PASSPHRASES

The first implementation of the DPS would be an increase of the typical password size of eight characters to something larger; here we consider a passphrase consisting of approximately forty spaces. A typical dictionary attack would be useless against a forty-character passphrase, since there are no forty-character words in the standard English dictionary. This in itself is not a new concept. Computer security experts have been proposing a shift to passphrase for some time [Ref. 17, 19].

The use of non-dictionary words and special characters has forced attackers to alter their methods and their tools. Now, instead of using a dictionary attack, they need to use every possible string over the chosen alphabet. Such a brute force attack, though taking longer to crack passwords, would eventually be successful. Nevertheless, simply increasing the alphabet size and increasing the password length will help in making the authentication process more secure. On the other hand, Moore's Law states that the speed of the microprocessors will double approximately every eighteen months to two years. As the processor speed increases so will the search speed. This leads us into the second part of the DPS, which is its increased use of cryptography.

Currently, all of the password-cracking tools used to attack the password file are based on knowledge of the encryption methods or hashing schemes that are implemented by the operating system. Tools such as "John the Ripper," a password "cracker" for Unix systems, and L0phtcrack and Nordhal-Hagen's Password recover tool for Microsoft operating systems use the password protection schemes to match hashes or encryptions to discover the passwords. Nordhal's Password recovery tool actually allows an attacker to change the password of a user. The tool creates a hash for the new password, using Microsoft's hashing scheme implementation, then inserts the new hash, overwriting the old hash, and so changing the password. This is all done from outside the Microsoft environment, consequently bypassing Microsoft's security and audit features.

B. DPS IDENTIFICATION AND AUTHENTICATION

The DPS still retains the standard Identification and Authentication (I&A) procedures. A user logs into a computer using his user id, which is unique. Along with the user id, the user responds with a forty-character passphrase. The user ID and pass phrase are passed to an I&A server (figure 5.1).

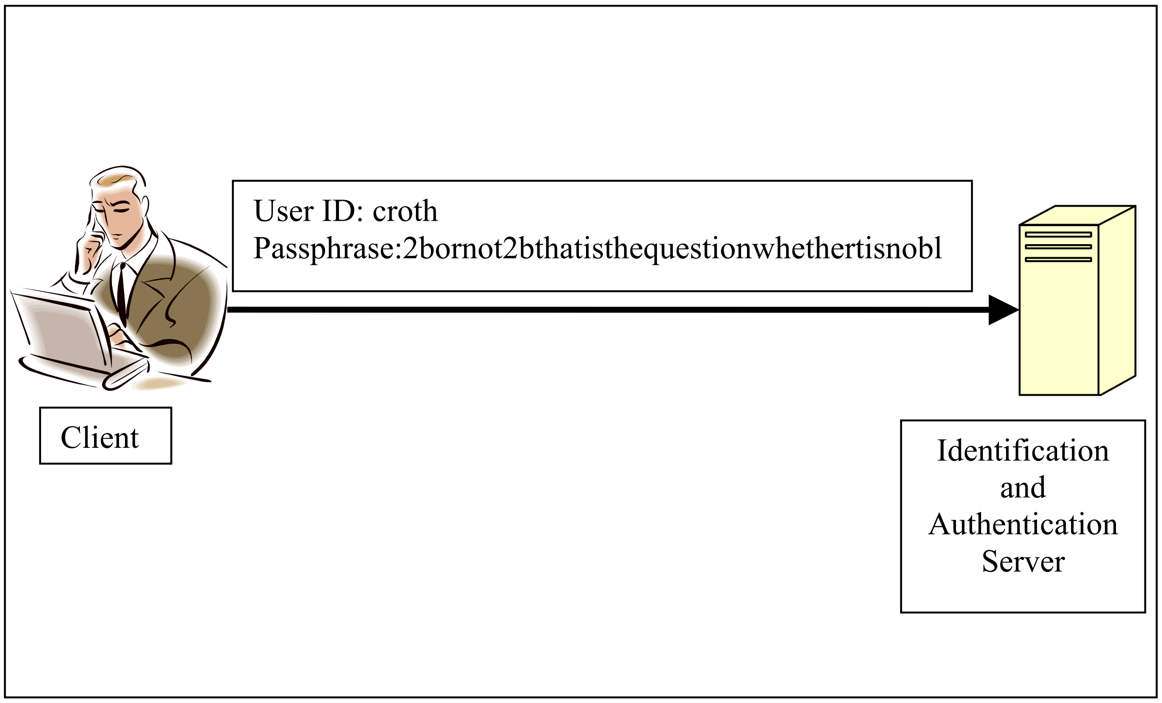

User ID: croth
Passphrase:2bornot2bthatisthequestionwhethertisnobl
Client
Identification and Authentication Server

Figure 5.1 Passing Credentials

The I&A server would first verify that the user id is valid by consulting a list on the server (Table 5.1). Associated with each valid user id would be a permutation of the set {1,2,3,4,5}. These permutations are themselves associated with 5!=120 different encryption methods. The user's passphrase is broken into five segments, and the encryption method applied to the *j*th segment is determined by the *j*th component in the associated permutation. The sequence of permutations would be "randomly" generated. After 120 users, a new sequence would begin.

USER ID	Encryption Sequence
croth	23145
bmichael	54123
ras	32514
tjdevlin	43215
aaharper	32541
…	…

Table 5.1 user_id Table

We will demonstrate this procedure using the forty-character passphrase, "2bornot2bthatisthequestionwhethertisnobl," and user id, "croth". We first break this phrase into five, eight-character segments (see table 5.2).

Segment 1	Segment 2	Segment 3	Segment 4	Segment 5
2bornot2	bthatist	hequesti	onwhethe	rtisnobl

Table 5.2 Segmented Passphrase

Each of these segments would then be passed from the I&A server to an encryption device, as determined by the table on the I&A server. We propose a separate encryption device, not co-located with the I&A server, for each segment. In this example, we would use five separate computers for encryption.

Returning to our example above, the user croth's pass phrase has been segmented. The I&A server passes each segment, along with the user id, to the assigned encryption device. Continuing with the example above, Segment 1 (2bornot2) would go to encryption device 2, Segment 2 (bthatist) would go to encryption device 3, Segment 3 (hequesti) would go to encryption device 1, and so on (figure 5.2).

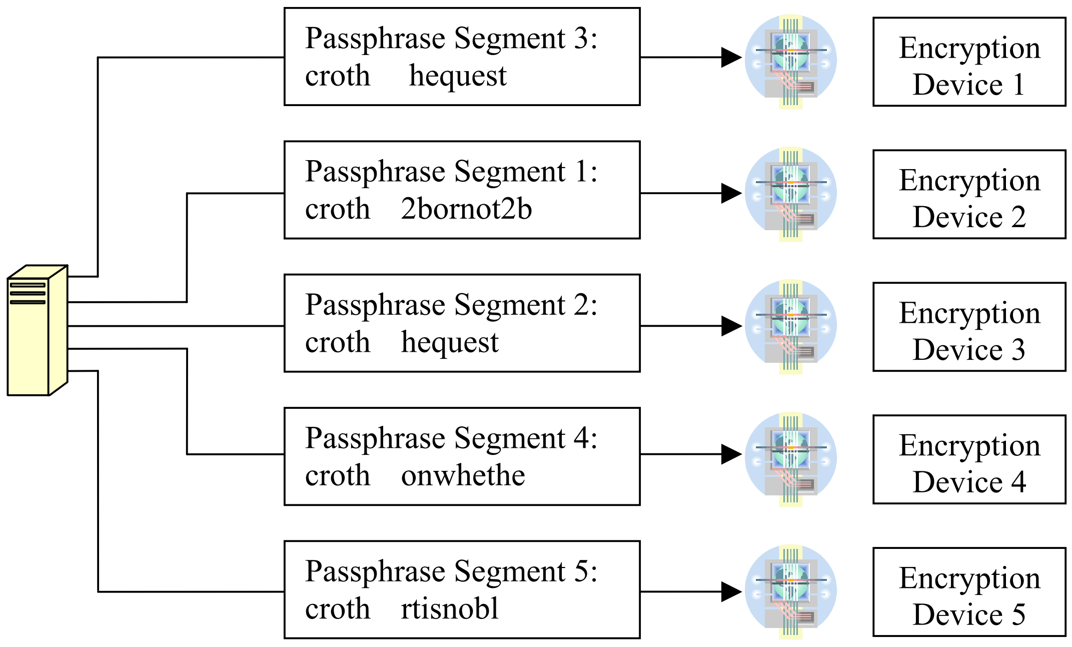

Passphrase Segment 3: croth hequest
Encryption Device 1
Passphrase Segment 1: croth 2bornot2b
Encryption Device 2
Passphrase Segment 2: croth hequest
Encryption Device 3
Passphrase Segment 4: croth onwhethe
Encryption Device 4
Passphrase Segment 5: croth rtisnobl
Encryption Device 5

Figure 5.2 Encryption of Segments

The I&A server would not do any of the encryption itself. The only information that it would store would be the user ID and the sequence. The only information that it would pass to each encryption device would be the user ID along with the assigned segment. At this point, the I&A server would wait for responses from the encryption devices, either confirming or denying that the segment is valid for this user.

C. ENCRYPTION PHASE

The next phase of the DPS is the encryption phase. The encryption itself is not the weakness that enables password cracking. Passwords are not reverse-engineered by inverting the hash or the encryption. Rather, the same encryption scheme used by the operating system under attack is used to encrypt or hash words or phrases in an attempt to match what is stored in the password file. What DPS proposes is the incorporation of multiple encryption schemes in the I&A process. By segmenting the passphrase, using multiple encryption methods, and storing the hashes separately, we mitigate the risks associated with a single password file.

1. Encryption Devices

Each of the encryption devices would receive its segment of the passphrase and the user id. The simplest way to understand the process is to compare each of the encryption devices, or servers, to an instance of the current password that holds only one fifth of the actual passphrase (figure 5.3).

After the initial installation, each of the encryption devices would have a particular encryption method installed. We propose to allow the option of using different encryption methods for each device. In fact, we would encourage this. The idea is to use encryption methods that have been accepted by cryptographers for use with passwords (e.g., DES, 3DES, MD5 hashing, Advanced Encryption Standard) in a multi-layer defense. Each device would have its own encryption scheme and a database containing the user id and cryptographic value of the input segment.

In our example, Segment 1 (2bornot2) would go to encryption device 2. Encryption Device 2 is using Data Encryption Standard (DES). Upon receipt, encryption device 2 would encrypt the segment and compare it to the value in the database. If the value in the database equals the encryption value, then this segment is accepted as valid.

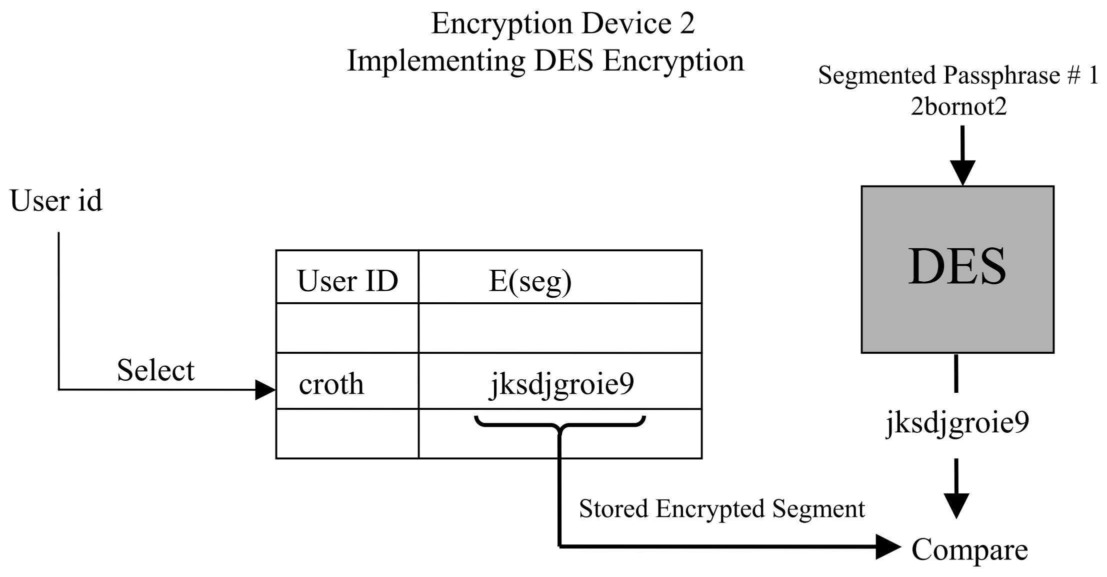


Figure 5.3 DES Encryption Example

The encryption device sends a validation back to the server for this segment (figure 5.4).

Each of the encryption devices would go through the same sequence of steps, returning a validation or denial to the I&A server. The difference would be that each encryption device would use a different encryption scheme. The encryption devices would not interact with one another, and each would be responsible only for its own segment of the passphrase.

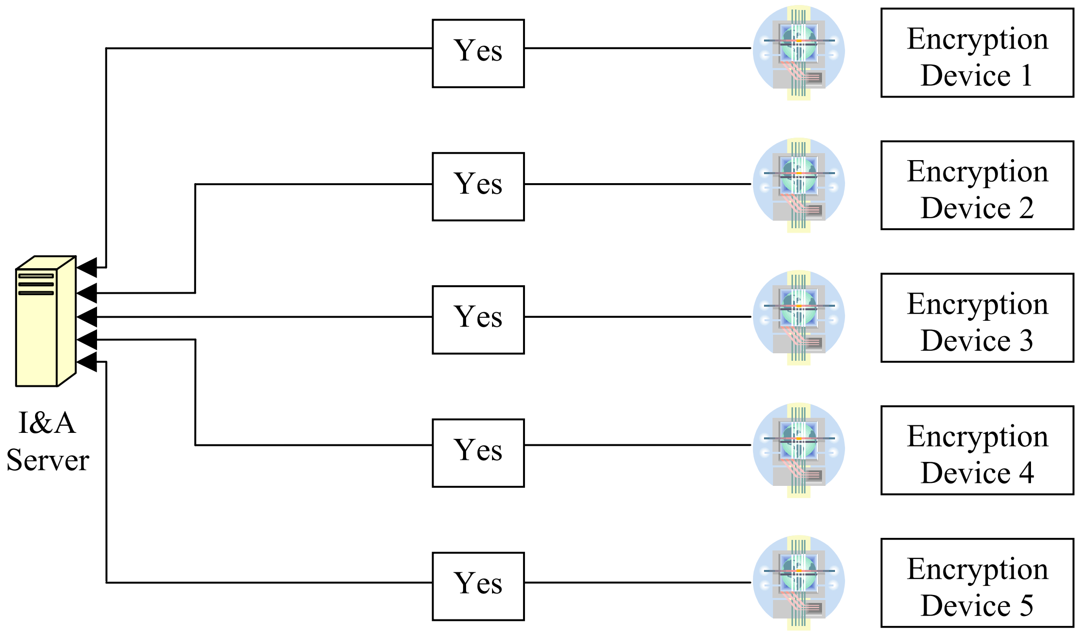


Figure 5.4 Responses Back to Server

Only after the I&A server has received all five validations, one from each encryption device, would the user be authenticated. If any response comes back negative, the user is denied access to the system.

Upon receipt of a full set of validations, the user is authorized access to the computer system.

D. ANALYSIS OF THE DPS ALGORITHM

After observing that a single password file is vulnerable and often exploited, we have informally outlined a possible solution, which is to distribute the password file. Whether the DPS is a viable scheme remains to be determined. We must ensure that the scheme will actually work, and that the additional complexity will be acceptable. To facilitate this analysis, we state the DPS procedure more formally.

```
Algorithm for LOGIN
id              string
phrase          string
read id
read phrase
IDENTIFY (id, phrase)
```

Upon user login, the system reads the user's credentials, id, and phrase. These credentials are passed to the Identification and Authentication server using the IDENTIFY procedure..

Algorithm for IDENTIFY

```
IDENTIFY (user_id, passphrase)          // user_id and passphrase read in by
                                        // login procedure

valid                           boolean
x,y, i, j,                      integers
user_id, passphrase, segment    string
sequence_num[ ]      array              //array of numbers no larger than five
                                        // elements consisting 1-5
user_table           record             //  Table consisiting of a user_id,
                                        //  with an associated five digit
                                        //  sequence number

valid ← true
x ← 1
y ← 8
i ← 1
j ← 0

if (SEARCH (user_table, user_id) not = 0)       // search returns seq_num[ ]
                                                // if user_id found
    then
        do while (valid and i < 6)              // sentinel control for while
                                                // loop breaks if any
                                                // authenticates return false
                                                // or if all five authenticates
                                                // return true
            segment ← PARTITION (passphrase, x, y)
            j ← sequence_num[i]
            i ← i + 1
```

```
                    x ← x + 8
                    y ← y + 8

                    if j = 1
                        then valid ← AUTHENTICATE_1 (user_id, segment)
                        else if j = 2
                            then valid ← AUTHENTICATE_2 (user_id,
                        segment)
                            else if j =3
                                then valid ← AUTHENTICATE_3 ….
                            …
                if (not valid) access denied
                    else if valid access granted
else
access denied
```

The algorithm calls the SEARCH routine to verify that the user_id exists in the user table. If the user_id exist, then segment each portion of the passphrase with the PARTITION procedure. Each segment is then paired with the value of the *i*th element of the sequence number, authenticated using the AUTHENTICATE_procedures. If any of the AUTHENTICATE procedures returns false, or if they all return true, control breaks from the while loop. A single failure of AUTHENTICATE denies user access to the system.

```
SEARCH (user_id)

sequence_num ← nil
count ← 0
    while count < size of user table
        do  count ← count +1
            if user_id = user_table [count]
                then   sequence_num ← sequence_table [count]
            else
return sequence_num
```

SEARCH searches the user_id_table, indexed on the user_id. If the user_id is found, then the sequence number is returned. If the user_id is not found (invalid user) then nil is returned.

```
PARTITION (passphrase, x, y)

segment string
segment ← copy passphrase starting at the xth character
    ending at the yth character
return  segment
```

PARTITION copies the passphrase starting at the xth character and ending at the yth character, creating an eight character segment. It then returns this segment

```
AUTHENTICATE
AUTHENTICATE (user_id, segment)
if SEARCH (user_id) not = 0                    // Searches encryption table
                                               // indexed on user. If user exist
                                               // return encrypt_tag_seg else return
                                               // 0
                                               // encrypted_table_segment
    then
        if ENCRYPT(segment) = encrypt_tab_seg
            then return true
        else
    return false                  // calls the preset encryption
                                               //routine
                                               // returns encrypted segment
else
return false.
```

AUTHENTICATE searches the encryption table for a valid user id. If the user id is found, the segment is encrypted and compared to the encrypt_tab_seg. If both these values are equal, the procedure returns true. Otherwise the procedure returns false. Since each AUTHENTICATE procedure is the same (except the actual ENCRYPT function) it only needs to be listed once.

The DPS does not use new storage, search or encryption techniques. The purpose of the DPS is to build on tools that are already available and in current use. Even the encryption will not add additional complexity because each encryption device is handling one segment and doing one set of calculations.. There are five segments that need to be encrypted, but each is being processed at a separate location in constant time. The complexity of encryption can therefore be reduced to a constant. It follows that the overall complexity of this authentication procedure is driven solely by the search implementation. Since the search algorithms for the sequence table and the encryption table are standard linear searches, with run time O(n) [Ref.8], our scheme requires O(n) time, where n is the number of entries in the user table. This complexity may be reduced further, if required, by implementing other search and storage algorithms.

E. UML MODEL

After determining that the DPS adds no significant computational complexity, the next step is an analysis of the design. The DPS is an ideal candidate for using an object-oriented approach. This allows us to model the DPS in the Unified Modeling Language (UML).

Analysis emphasizes the investigation of the problem rather than definition of a solution [Ref. 22]. In the DPS system, we have determined that the problem in current computer systems is the implementation of a single password file and the vulnerabilities associated with this file.

The design phase emphasizes a high-level and detailed description of a logical solution and the way in which it fulfills the requirements and constraints in effect [Ref. 22]. Using UML, we can finish both the analysis and the design phase in an object-oriented analysis and design. This will allow ease of transition during the construction or object oriented programming, because the design components could than be implemented in such object oriented languages as C++, Java, Smalltalk, or Visual Basic, to name just a few [Ref. 22, 23].

We can easily describe the DPS using UML Use Cases. Table 5.3 and 5.4 show the high level Use Case and the expanded level Use Case for the DPS. Specifically, they describe the user login procedures.

Use Case	**Login**
Actors:	User
Type:	Primary (to be discussed)
Description:	User attempts to login into a computer system. The user passes credentials to the system for Identification and Authentication. Upon acceptance of credentials, user access the system

Table 5.3 High Level Use Case

The expanded case allows us to show more detail, and essentially obtain a deeper understanding of the DPS process and requirements.

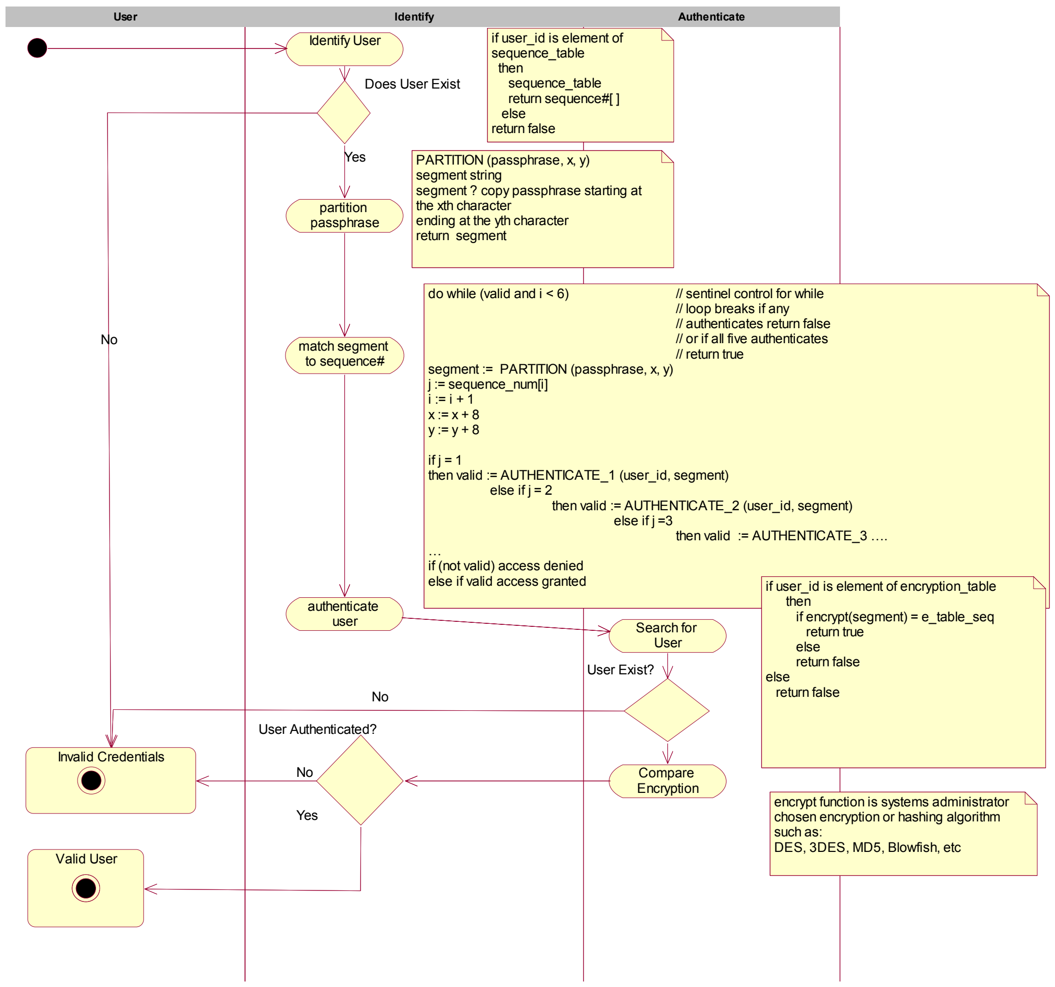
Use Case:	User Login
Actors:	User
Purpose:	Identify and Authenticate user to the network or system
Overview	A user logs into a client workstation presenting a user id and passphrase. The I&A server checks the user id, then partitions the passphrase into five segments. Once each segment is encrypted and verified, the user is authorized access to the system.
Type	Primary and Essential

Typical Course of Events	
Actor Action	System Response
1. This use case begins when a user tries to access the system	
2. The user presents credentials: user_id, passphrase	3. Determine whether user_id is valid by searching user_id table
	4. If user_id is valid, partition passphrase, and pass segments to identified encryption device, according to sequence.
	5. Receive responses back from encryption devices. If all responses positive allow user access to system.
6. User Access System	

Table 5.4 Expanded Use Case

The Use Case allows for us to map the sequential flow of activities into the activity diagram. Activity diagrams provide a way to model the workflow of a business process, or in our case the Identification and Authentication process for the DPS. This also allows us to model code-specific information such as a class operation for easier transition to an object oriented language.

The activity diagram allows us to model the DPS workflow (Fig. 5.6). The transition of passphrase from user to the I&A server and its subsequent segmenting allows for easier understanding from a design point of view. The notes to the side for pseudocoding allow for an easier implementation in coding the DPS in an object oriented language. The activity diagram would assist as the model in the software development process. [Ref. 23].



```
if user_id is element of
sequence_table
  then
     sequence_table
     return sequence#[ ]
   else
return false
```

```
PARTITION (passphrase, x, y)
segment string
segment ? copy passphrase starting at
the xth character
ending at the yth character
return segment
```

```
do while (valid and i < 6)                    // sentinel control for while
                                              // loop breaks if any
                                              // authenticates return false
                                              // or if all five authenticates
                                              // return true
segment := PARTITION (passphrase, x, y)
j := sequence_num[i]
i := i + 1
x := x + 8
y := y + 8

if j = 1
then valid := AUTHENTICATE_1 (user_id, segment)
           else if j = 2
                      then valid := AUTHENTICATE_2 (user_id, segment)
                                 else if j =3
                                            then valid  := AUTHENTICATE_3 ....
...
if (not valid) access denied
else if valid access granted
```

```
if user_id is element of encryption_table
    then
       if encrypt(segment) = e_table_seq
          return true
       else
       return false
else
   return false
```

```
encrypt function is systems administrator
chosen encryption or hashing algorithm
such as:
DES, 3DES, MD5, Blowfish, etc
```

Figure 5.5 Activity Diagram

The sequence diagram provides a graphical view of a scenario that shows object interaction in a time-based sequence. In the DPS, the sequence diagram provides us the necessary clarification of each phase of the login process [Ref. 23].

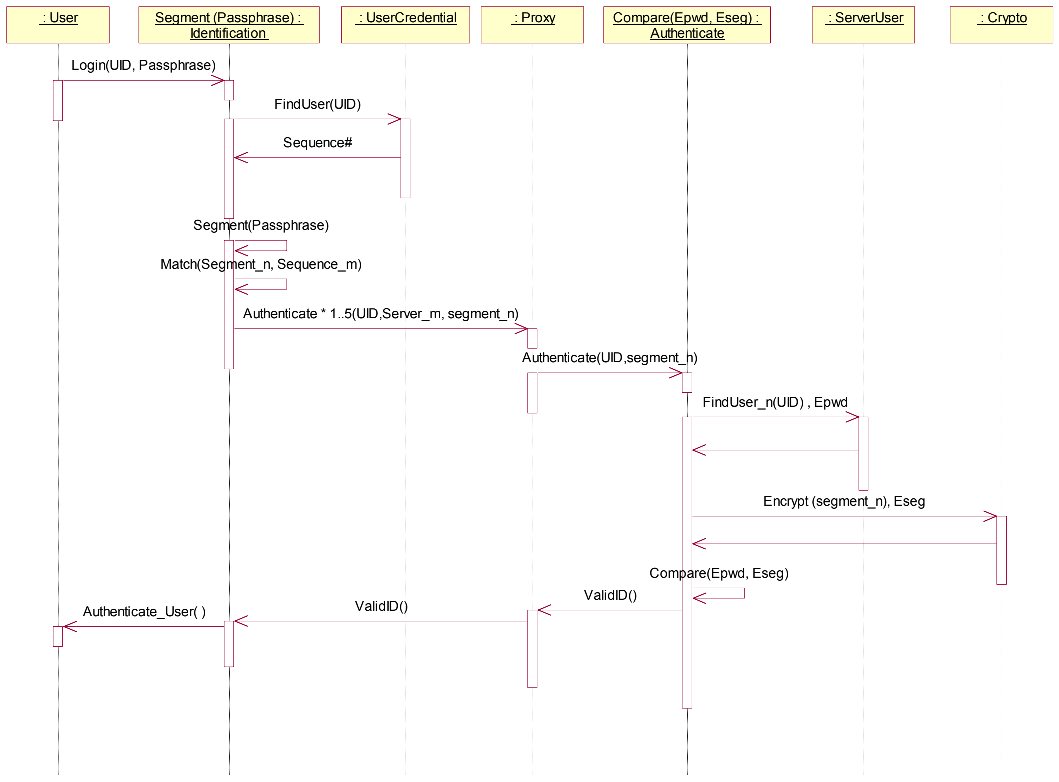

: User
Segment (Passphrase) : Identification
: UserCredential
: Proxy
Compare(Epwd, Eseg) : Authenticate
: ServerUser
: Crypto
Login(UID, Passphrase)
FindUser(UID)
Sequence#
Segment(Passphrase)
Match(Segment_n, Sequence_m)
Authenticate * 1..5(UID,Server_m, segment_n)
Authenticate(UID,segment_n)
FindUser_n(UID) , Epwd
Encrypt (segment_n), Eseg
Compare(Epwd, Eseg)
ValidID()
ValidID()
Authenticate_User( )

Figure 5.6 DPS Sequence Diagram

Modeling in UML allows us to quickly see that the implementation of the DPS is not difficult. We have clearly defined the boundaries and procedures for each phase of the login procedure of the DPS. It also allows us to track the flow control of the program. We could further refine these in defining a state diagram for further security analysis. Use of a software tool such as Rational Rose would help us in porting the DPS into an actual object oriented program [Ref. 23].

For a pilot implementation of the DPS, we turned our attention to modification of a current operating system.

F. LINUX: A CANDIDATE FOR THE DPS

After a review of operating systems and source code we decided that the ideal candidate for a first implementation of the DPS would be a Linux-based operating system that supports the Pluggable Authentication Modules (PAM). These operating systems include Caldera, Debian, Red Hat Linux, SuSe Linux, and MSC.Linux. Even Apple OS-

X has implemented Linux-PAM [Ref. 24]. Linux-PAM is a suite of shared libraries that enable local systems administrators to choose how applications authenticate users [Ref. 24]. PAM allows the system administrator to set authentication policies for PAM-aware applications without having to recompile authentication programs. PAM does this by utilizing a pluggable, modular architecture. The precise modules PAM calls for a particular application are determined by looking at that application's PAM configuration file in the */etc/pam.d* directory [Ref. 24]. A Linux-PAM module is a single executable binary file that can be loaded by the Linux-PAM interface library.

Currently PAM is employed by the I & A process for managing password security. Tasks include enforcing policies regulating length and maximum age of passwords, tracking the changes users make to their passwords, and a host of other functions to deter password cracking. However, the real strength in PAM is its flexibility in the authentication process.

PAM gives systems administrators the ability to choose an authentication scheme. PAM allows authentication processes to range anywhere from voice recognition to one-time passwords. It does this by separating the I & A process into four types of management tasks: authentication management, account management, session management, and password management. This process is all modular, allowing modules to be stacked upon one another. The use of these modules enables PAM to search through several different password databases [Ref. 24]. For example, the Apache web server has a module that provides PAM services. This allows additional operating system password and protection schemes to be used. There are PAM modules that allow the use of series of databases to authenticate users. This allows authentication using LINUX based password databases in conjunction with password databases such as those from NT, or Novell. A systems administrator can create an authentication process, for a particular system in conformance with the PAM specifications, and then implement it without modifying any of the applications on the system. The administrator can even incorporate current PAM processes without rewriting or recompiling these PAM-aware applications [Ref. 25].

A possible DPS PAM-aware application might look like this:

```
1       #%PAM-1.0
2       auth        required      /lib/security/pam_dps.so userid
3       account     required      /lib/security/pam_segment.so
4       password    required      /lib/security/pam_encrypt1.so
5       password    required      /lib/security/pam_encrypt2.so
6       password    required      /lib/security/pam_encrypt3.so
7       password    required      /lib/security/pam_encrypt4.so
8       password    required      /lib/security/pam_encrypt5.so
9       session     required      /lib/security/pam_unix.so
```

The first line is a comment line. The second line calls the module that prompts for a user name and passphrase. It checks the user_id for validity using the information stored in the */user_id* file. If the user exists, the passphrase is then passed to the segment module. This breaks the passphrase into five separate segments. It then sends each of the segments to their respective encryption module. The final line specifies that the session component of the pam_unix_so module will manage the session.

As stated previously, the real strength in PAM is its flexibility. PAM modules that are already created could be incorporated into any PAM aware application. We could theoretically replace line 4 with

```
auth    required            /lib/security/pam_unix.so,
```

allowing use of the standard Unix password scheme, which stores the password in the */etc/passwd*. We could also incorporate additional security modules such as the pam_cracklib.so to see if a segment can be easily determined by a dictionary-based password-cracking program.

By stacking the modules, we force each of the encryption modules to return positive responses before allowing a user session access.

G. MONETARY COST OF IMPLEMENTATION

The initial design layout of the DPS consists of a client, an I&A server, and five encryption devices. The physical model cost would consist of the expenses related to the I&A server and the five encryption devices. Each encryption device would consist of a standalone computer system. Given this broad description, and without including cost of programming, our cost could run to several thousands of dollars.

One strategy by which we might reduce the cost would be to implement the DPS in a Linux Beowulf Cluster, which allows for the implementation of a master computer and slave computers. The slave computers are basically processors equipped with storage. This approach would eliminate the cost of monitors and keyboards for the encryption devices.

Another possibility is the implementation of virtual machines on the I&A server itself. This would eliminate the need for additional computers altogether. VMware is a perfect example of a software solution of this kind. It allows us to operate several guest operating systems within one host operating system. These guest operating systems are given separate disk space as well as their own Internet Protocol (IP) addresses. In essence, one can create a virtual network on a single machine. This allows us to have one physical machine, that is logically an I&A server with five encryption devices (fig. 5.8). Each of the encryption databases would be accessible only through the guest operating system. Even if the host system were compromised, the guest system would not necessarily be. The cost of this would consist only of the software license, approximately three hundred dollars.

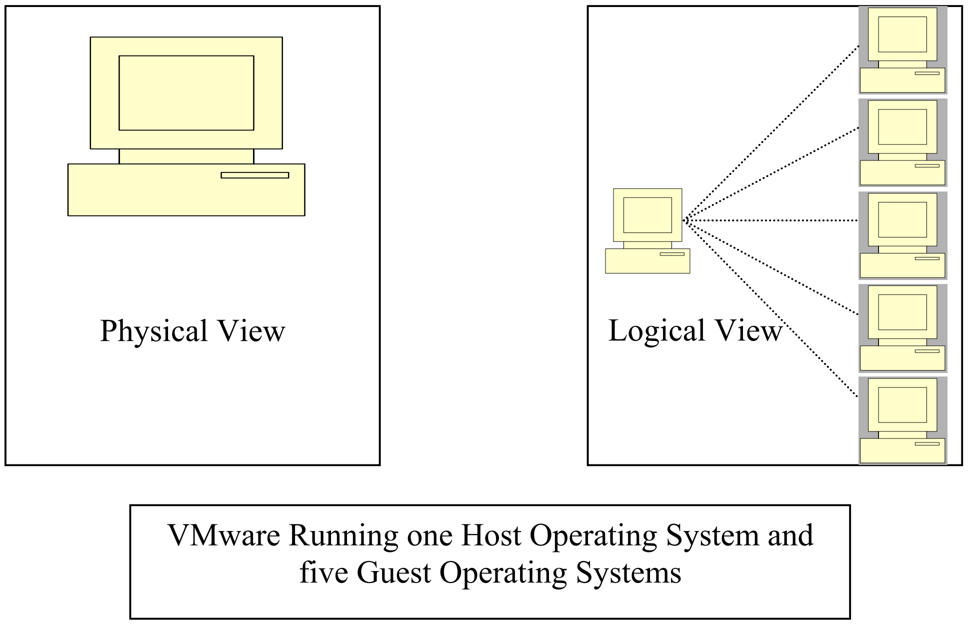

Physical View
Logical View
VMware Running one Host Operating System and five Guest Operating Systems

Figure 5.7 Physical and Logical View VMWare

H. WEAKNESSES OF THE DPS

The DPS was introduced to counter the weaknesses of passwords and the single password file. The DPS does not eliminate all security concerns. It also has some weaknesses.

1. Denial of Service

By introducing five segmented passphrases, the system relies on the response of all five segments before it grants authorization to a user access. If any of these segments returns with a negative reply then user access is denied. The system is vulnerable to the possibility of a denial of service if any one of the five segments is prevented from encrypting its segment. The system is also vulnerable if any of the encryption devices is blocked from sending a positive response. If this happens, the system is unavailable to a legitimate user. This violates one of the tenets of computer security, namely availability of the system.

We might address this by creating additional encryption devices serving as backup devices. This solution would have an increased monetary cost for the additional hardware to implement this solution. There could be additional computational complexity cost as the algorithm would have to be rewritten to accommodate additional devices, as well as the timing, sequencing, and selection of the devices were incorporated into the system.

2. Software Implementation

Implementing the encryption in a software device gives us the advantages of flexibility and portability, ease of use, and ease of upgrade [Ref. 20]. The ability to choose the encryption scheme per device is one of the biggest advantages of the DPS. It might also be its weakness. The encryption algorithm could be replaced with a weak or reversible algorithm. The management of the keys, in this case the password segments, must be secured. The segments should not be stored on disk or written to a place in memory.

The speed and cost of software implementation of encryption is another disadvantage. Standard DES and RSA encryption, run inefficiently on general purpose processors. We propose moving each encryption to a separate processor, that only handles the encryption to counter this problem. Even though some cryptographers have tried to make their algorithms more suitable for software implementation, specialized hardware will always be faster [Ref. 20].

An encryption algorithm running on a generalized computer has no physical protection, where hardware encryption devices can be securely encapsulated to prevent physical tampering [Ref. 20].

Hardware implementation of cryptography is also easier to install than the corresponding software version. It is cheaper to put special-purpose encryption in hardware, than it is to put it in a microprocessor and software [Ref. 20]. Even when encrypted data comes from the computer, it is easier to install a dedicated hardware encryption device than it is to modify the computer's systems software. The only way to make encryption invisible to the user in software is to bury it deep inside the operating system, which is not easy [Ref. 20].

3. Mistyped Passphrase

Though there are no statistics to track how many times users mistype their passwords, a longer passphrase would probably increase the number of failed login attempts simply because of its length.

4. Common Passphrases

The potential problems of common passphrases still remain. One such problem would be the use of personal information (e.g., a father using all of his children's names). An attacker using social engineering skills would still be capable of discovering the passphrase, as the previously-mentioned study suggests. Another potential problem could be the increase in the small number forty-character pass phrases that may not be in the dictionary but that are nevertheless part of modern literature (e.g., Supercalifrajalisticexpialidocious!!!!!!) The more cumbersome authentication method also increases the risk of a user writing down the passphrase, or, if the DPS is implemented on multiple systems, using the same phrase more than once.

5. Login Delays

The added procedures for parsing the passphrase into segments, and encrypting each segment do not add additional delays to any one section of the DPS. However, the user is not authenticated until all encryption servers return a positive response. Awaiting five responses might add additional time to the authentication of a user

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATION FOR FUTURE WORK

Passwords and password files continue to be weaknesses in the Identification and Authentication. There is a need to address this weakness because it has been identified as a consistent point of failure. In this thesis we described how current computer systems incorporate passwords into their I&A process. These systems included open-source UNIX-type operating systems as well as commercial off the shelf products for which the source code is not freely available, such as Microsoft's network operating systems. The weaknesses and commonly exploited vulnerabilities were analyzed to facilitate development of a solution that would not add significant cost to the end user.

In particular, the work described in this thesis models a Distributed Password Scheme (DPS) in a network environment. The DPS proposed replaces the eight-character password with a forty-character passphrase, segments this passphrase, distributes the various segments among subsystems, and incorporates multiple encryption techniques for protecting these distributed segments. The advantages of such an implementation are the elimination of a single "hackable" password file, the elimination of easily-guessed common passwords, and resistance to current "hacker tools".

The thesis presented a solution that did not add significant amount of computational complexity to current systems, while incorporating current available technology and approved cryptography. The model was further designed and refined using the Universal Markup Language (UML).

The model is a reference for implementing an I&A process that does not depend on the single password file. The model has not been finalized or perfected; this leaves a number of targets for future research. The next step would be the actual implementation of the DPS.

In considering such an implementation, Linux-Pluggable Authentication Modules (Linux-PAM) were viewed as ideal candidates for the DPS. Linux-PAM allows for the incorporation of various authentication methods without creating significant cumbersome changes to the operating system or current applications. It also allows incorporation of current Linux-PAM security measures for strengthening the I&A process.

Actual implementation of a software solution in a network environment would allow the collection of data in a real-time environment. This would allow the analysis of latency problems during transmission and storage of passwords and passphrases in memory as well as the protection of the databases in memory and secondary storage. Implementation costs were also discussed, including a possible cost-efficient prototype solution using VMWare on a single computer. Future research is needed to study the feasibility and scalability aspects that result in a software design. Future research can determine the extent and limitations of a software design, as well as a hardware implementation. Possible research in analyzing both implementations could be done.

The DPS is not presented as a “silver bullet” solution to computer security. The thesis presented a solution to address the single password file weakness. Implementing the DPS adds another layer of protection but in no way are we touting this as the computer security solution.

LIST OF REFERENCES

[1] Pfleeger, Charles P. (1997). *Security in Computing*. Upper Saddle River, NJ: Prentice Hall.

[2] Squitieri, Tom. (2002, May 5). Cyberspace Full of Terror Targets. *USAToday*.

[3] Freedman, David H., and Mann, Charles C. (1997). *@Large, the Strange Case of the World's Biggest Internet Invasion*. New York, NY: Simon and Shuster.

[4] Hafner, Katie and Markoff, John. (1991). *Cyberpunk, Outlaws and Hackers on the Computer Frontier.* New York, NY: Touchstone, 1991.

[5] Stoll, Clifford. (1989). *The Cuckoo's Egg*. New York, NY: Pocket Books.

[6] Lemos, Robert. (3 May 2002). Hacker Finds Fault in .NET Security. *C/Net News.com*,.http://news.com.com/2100-1001-898219.htm.

[7] Scambray, Joel, and McClure, Stuart. (2001). *Hacking Exposed Windows 2000: Network Security Secrets & Solutions*. New York, NY: Osborn/McGraw-Hill.

[8] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. (2001). *Introduction to Algorithms, Second Edition*. Cambridge, MA: MIT Press.

[9] Stallings, William. (1999). *Cryptography and Network Security, Principle and Practice*. Upper Saddle River, NJ: Prentice Hall, Inc.

[10] Hatch, Brian, Lee, James, and Kurtz, George. (2001). *Hacking Linux Exposed: Linux Security Secrets & Solutions*. New York, NY: Osborn/McGraw-Hill.

[11] Jackson, Michael. (1996). Linux Shadow Password HOWTO. http://www.tidp.org/HOWTO/Shadow-Password-HOWTO.html#2toc2, April 1996.

[12] Morris, R., and Thompson, K. (1979). Password Security: A Case History. In Comm ACM, v22 n 11, Nov 1979.

[13] Kolde, Jennifer. (2000). Password Security in Windows NT and Windows 2000. http://www8.wire.com/articles/print_article.asp?printAID=1582, December 2000.

[14] Denning, Dorothy. (2000). *Information Warfare and Security*. Reading, MA: Addison-Wesley.

[15] Reynolds, James. (2001). Mac OS X Security - Password file. http://www.macos.utah.edu/Documentation/Mac_OS_X_Security/passwd.html. June 6 2001.

[16] Meinel, Carolyn. (2000). *Uberhacker! How to Break into Computers*. Port Townsend, WA: Loompanics Unlimited.

[17] Schwartau, Winn. (2000). *Cybershock, Surviving Hackers, Phreakers, Identity Thieves, Internet Terrorists, and Weapons of Mass Disruption*. New York, NY: Thunder's Mouth Press.

[18] Brown, Andrew. (2002, March 13). UK Study: Passwords Often Easy to Crack. CNN.com/SCI-TECH. http://www.cnn.com/2002/TECH/ptech/03/13/dangerous.passwords/index.html?related.

[19] Schneier, Bruce. (2000). *Secrets & Lies: Digital Security in a Networked World.* New York, NY: John Wiley & Sons, Inc,.

[20] Schneier, Bruce. (1996). *Applied Cryptography Second Edition: Protocols, Algorithms, and Source Code in C*. New York, NY: John Wiley & Sons, Inc.

[21] McNamara, Joel. (1997, August). Practical Attacks on PGP, http://www.privacy.com.au/pgpatk.html.

[22] Larman, Craig. (1998). *Applying UML and Patterns, an Introduction to Object-Oriented Analysis and Design*. Upper Saddle River, NJ: Prentice Hall PTR..

[23] Quantrani, Terry. (2000). *Visual Modeling with Rational Rose 2000 and UML*. Boston, MA: Addison-Wesley.

[24] Morgan, Andrew. (2002, March). Linux-PAM Users. http://www.kernel.org.

[25] -http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manuals/ref-guide/s1-pam-modules.html (2000, June).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Lieutenant Commander Chris Eagle
   Naval Postgraduate School
   Monterey, California

4. Professor James B. Michael
   Naval Postgraduate School
   Monterey, California

5. Professor Craig Rasmussen
   Naval Postgraduate School
   Monterey, California

6. Mr. Steve LaFountain (C4)
   National Security Agency
   Fort Meade, Maryland

7. Major Christopher Roth
   United States Army Student Detachment
   Fort Jackson, South Carolina